Лабораторна робота №7

Програмування операцій ділення чисел

Мета: Навчитися програмувати на асемблері ділення чисел, вивчити перетворення з двійкової у десяткову систему числення.

Завдання:

1. Створити у середовищі MS Visual Studio проект з ім'ям **Lab7**.
2. Написати вихідний текст програми згідно варіанту завдання. У проекті мають бути три модуля на асемблері:
   - головний модуль: файл **main7.asm**. Цей модуль створити та написати заново, частково використавши текст модуля main5.asm попередньої роботи №5;
   - другий модуль: модуль **module** попередньої роботи №6;
   - третій модуль: модуль **longop** попередньої роботи №6.
3. Додати у модулі процедури, які потрібні для виконання завдання. Обгрунтувати розподіл процедур по модулям.
4. У цьому проекті кожний модуль може окремо компілюватися.
5. Скомпілювати вихідний текст і отримати виконуємий файл програми.
6. Перевірити роботу програми. Налагодити програму.
7. Отримати результати – кодовані значення чисел згідно варіанту завдання.
8. Проаналізувати та прокоментувати результати, вихідний текст та дизасембльований машинний код програми.

Теоретичні відомості

Цілочисельне ділення

Нехай дані цілі числа *A* та *B*. Результатом ділення *A/B* може бути ціле або дробове число. Цілочисельним діленням є представлення результату ділення у вигляді двох цілих чисел: часткового *D* та залишку *R*. Запишемо це наступним чином:

$$\frac{A}{B} = D + \frac{R}{B} \qquad (1)$$

де: *D* – ціла частка результату, *R/B* – дробова частка результату.
Значення залишку $R = A \bmod B$, тобто *R* може бути від 0 до *B*-1.

Яка розрядність потрібна для представлення результатів ділення? Будемо розглядати ділення у двійковій системі числення. Введемо позначення: n_A, n_B, n_D, n_R – розрядність (кількість бітів) відповідно для *A, B, D* та *R*.

Якщо ділене *A* представлене n_A – бітовим двійковим кодом, то для часткового (*D*) потрібно бітів від (n_A - n_B +1) до n_A у залежності від значення дільника *B*. Наприклад, якщо n_A = 16, то максимальне значення для *A*:

A = 1111111111111111

Якщо n_B = 16, то у випадку максимального 16-бітового значення *B*

B = 1111111111111111

результатом ділення

$$\frac{A}{B} = \frac{1111111111111111}{1111111111111111} = 1$$

буде значення *D* = 1, для представлення якого достатньо одного біту – іншими словами: $n_D = n_A - n_B + 1 = 16 - 16 + 1 = 1$.

Якщо у двійковому коді *B* буде декілька, наприклад, п’ять старших нулів:

$$\frac{A}{B} = \frac{1111111111111111}{0000011111111111} = 100000\ (D)$$, залишок *R* = 11111

то фактично ділення не на 16-бітове, а на 11-бітове число, тому можна вважати n_B = 11. Результат (*D*) є 6-бітовим числом: $n_D = n_A - n_B + 1 = 16 - 11 + 1 = 6$.

Ще приклад:

$$\frac{A}{B} = \frac{1111111111111111}{0000010000000000} = 111111\ (D)$$, залишок *R* = 1111111111

Загалом, для представлення залишку (*R*) достатньо $n_R = n_B$ розрядів, оскільки *R* не може бути більшим, аніж *B*-1.

У випадку мінімального ненульового значення *B*=1 результат ділення

$$\frac{A}{B} = \frac{1111111111111111}{0000000000000001} = 1111111111111111$$

буде *D*=*A* із відповідною розрядністю $n_D = n_A$.

Алгоритм ділення цілих чисел "у стовпчик"

Розглянемо виконання операції $D = A/B$, де A та B цілі двійкові числа без знаку. Одним з найпростіших алгоритмів є ділення "у стовпчик". Нехай A=110101110101 та B=1010. Алгоритм ділення n-бітового числа A на 4-бітове B можна записати так:

1. Встановити $i=n$-4.
2. Взяти чотири старші біти числа A. Позначити це як $R=\{a_{n-1}, a_{n-2}, a_{n-3}, a_{n-4}\}$
3. Якщо R більше, або дорівнює B,
   то: i-та цифра (біт) результату d_i дорівнює 1, віднімаємо $R = R - B$
   інакше: i-та цифра результату d_i дорівнює 0
4. Зменшити i на одиницю. Якщо i менше нуля, то кінець роботи.
5. Помножити R на два. Таке множення означає зсув бітів на одну позицію вліво. Додати до R у молодший розряд i-й біт числа A, тобто a_i.
6. Перехід на п. 2.

У результаті роботи алгоритму отримаємо біти D та залишку R (рис. 1):

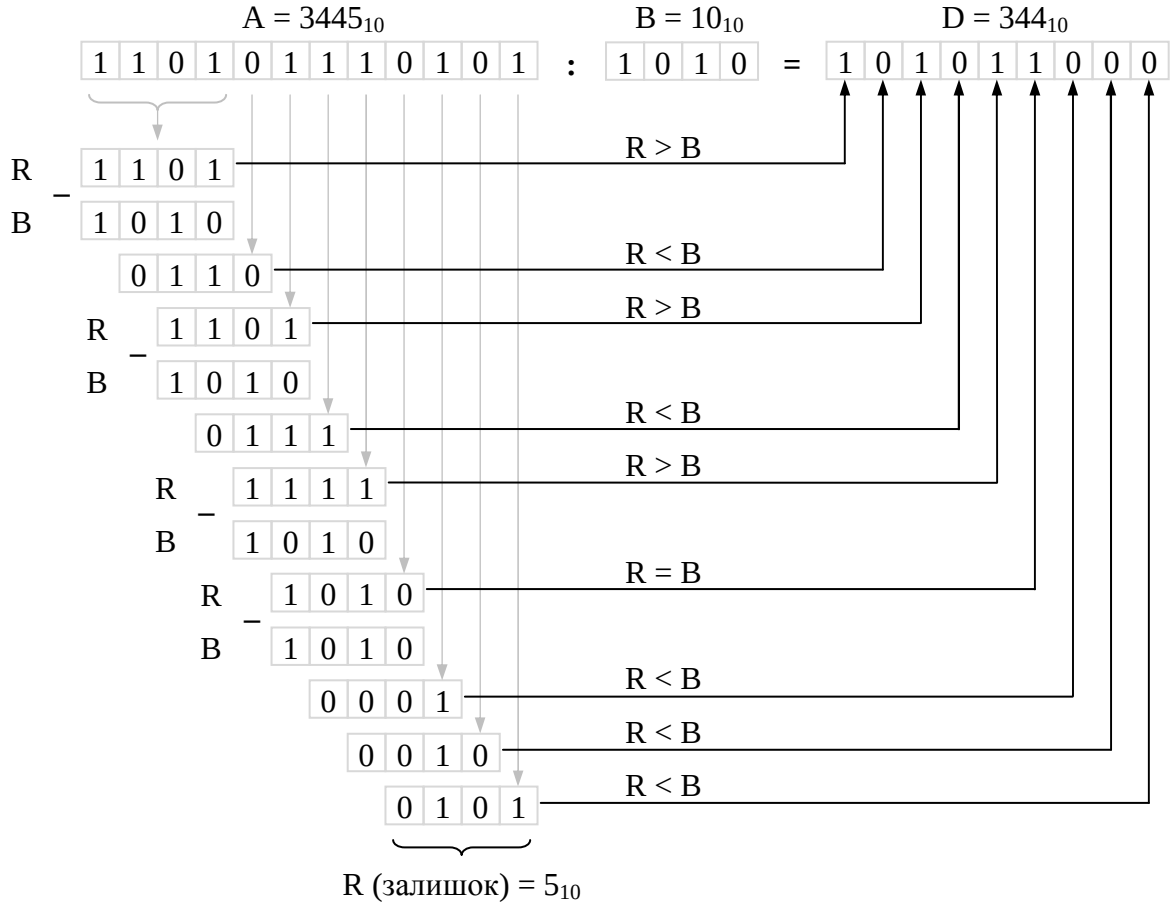


Рис. 1. Приклад ділення двійкових чисел

Алгоритм ділення групами бітів

Якщо у системі команд процесора є команди ділення, то можна побудувати алгоритм ділення підвищеної розрядності на основі послідовності таких команд. Іншими словами – групами бітів. Різновид алгоритму ділення D = A/B, який розглядається нижче, ґрунтується на тому, що багаторозрядне ділиме (A) у позиційній системі числення – у тому числі й у двійковому коді, може бути записане у вигляді суми груп розрядів. Розглянемо алгоритм ділення 16-бітового числа A на 8-бітове число B.

16-бітове число A можна представити у вигляді суми двох 8-бітових груп A_1 та A_0. Якщо потрібно, щоб результат ділення (D) був також 16-бітовим (наприклад, щоб запобігти переповнення розрядної сітки при B=1), то додамо зліва ще одну фіктивну 8-бітову групу A_2 з усіх нулів – число A від цього не зміниться (рис.2):

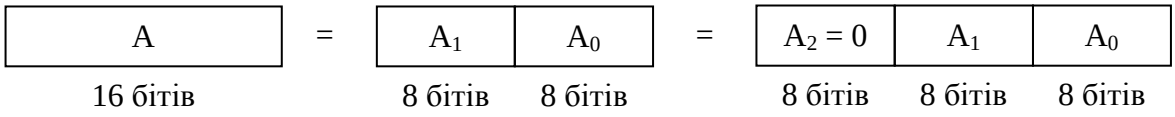


Рис. 2. Групи по вісім бітів числа A

Математично операцію ділення D = A/B можна записати у наступному вигляді:

$$D = \frac{2^{16}A_2 + 2^8A_1 + A_0}{B} = 2^8 \frac{2^8A_2 + A_1}{B} + \frac{A_0}{B}$$

Можна уявити собі пристрій, який ділить 16-бітове число на 8-бітове число. Результат цілочисельного ділення цілих чисел без знаку буде у вигляді двох 8-бітових цілих чисел: часткового D та залишку R (саме так працює один з різновидів команди DIV). Виконаємо ділення окремо двох старших груп розрядів числа A на B:

$$\frac{2^8A_2 + A_1}{B} = D_1 + \frac{R_1}{B}$$

Підставимо це у наведену вище формулу для ділення:

$$D = \frac{2^{16}A_2 + 2^8A_1 + A_0}{B} = 2^8 \left(D_1 + \frac{R_1}{B}\right) + \frac{A_0}{B} = 2^8 D_1 + \frac{2^8 R_1 + A_0}{B}$$

Тепер виконаємо ділення 16-бітового числа, старші 8-бітів якого є залишком R_1, а молодші – A_0 :

$$\frac{2^8 R_1 + A_0}{B} = D_0 + \frac{R_0}{B}$$

Загальну формулу ділення можна записати у вигляді:

$$D = \frac{2^{16}A_2 + 2^8 A_1 + A_0}{B} = 2^8 D_1 + D_0 + \frac{R_0}{B}$$

Таким чином, отримаємо результат ділення у вигляді 16-бітового числа, яке складається з 8-бітових D_1 та D_0 , а також 8-бітового залишку R_0. Величини D_1, D_0 та R_0 отримані послідовним діленням груп розрядів дільника A, починаючи зі старших розрядів.

Можна узагальнити цей алгоритм для ділення N-бітового числа *A* на K-бітове число *B* послідовними кроками ділення групами по K-бітів. Такий алгоритм можна назвати як ділення групами бітів "N:K".

Розглянемо алгоритм ділення підвищеної розрядності групами бітів "N:32". Для операції D = A/B розрядність ділимого (A) нехай буде N=128, а дільник (B) 32-бітовий. Число A складається з чотирьох 32-бітових груп розрядів. Можна записати, що

$$D = \frac{2^{96}A_3 + 2^{64}A_2 + 2^{32}A_1 + A_0}{B}$$

Таку операцію можна представити чотирма кроками 32-бітного ділення A_i на B. На кожному кроці виконується цілочисельне ділення (DIV) двійкового 64-бітового числа на 32-бітове число. Результат кожного кроку у вигляді 32-бітового часткового D_i та 32-бітового залишку R_i . Кінцевий результат ділення записується 32-бітовими групами D_i , а також 32-бітовим залишком R_0 (рис. 3).

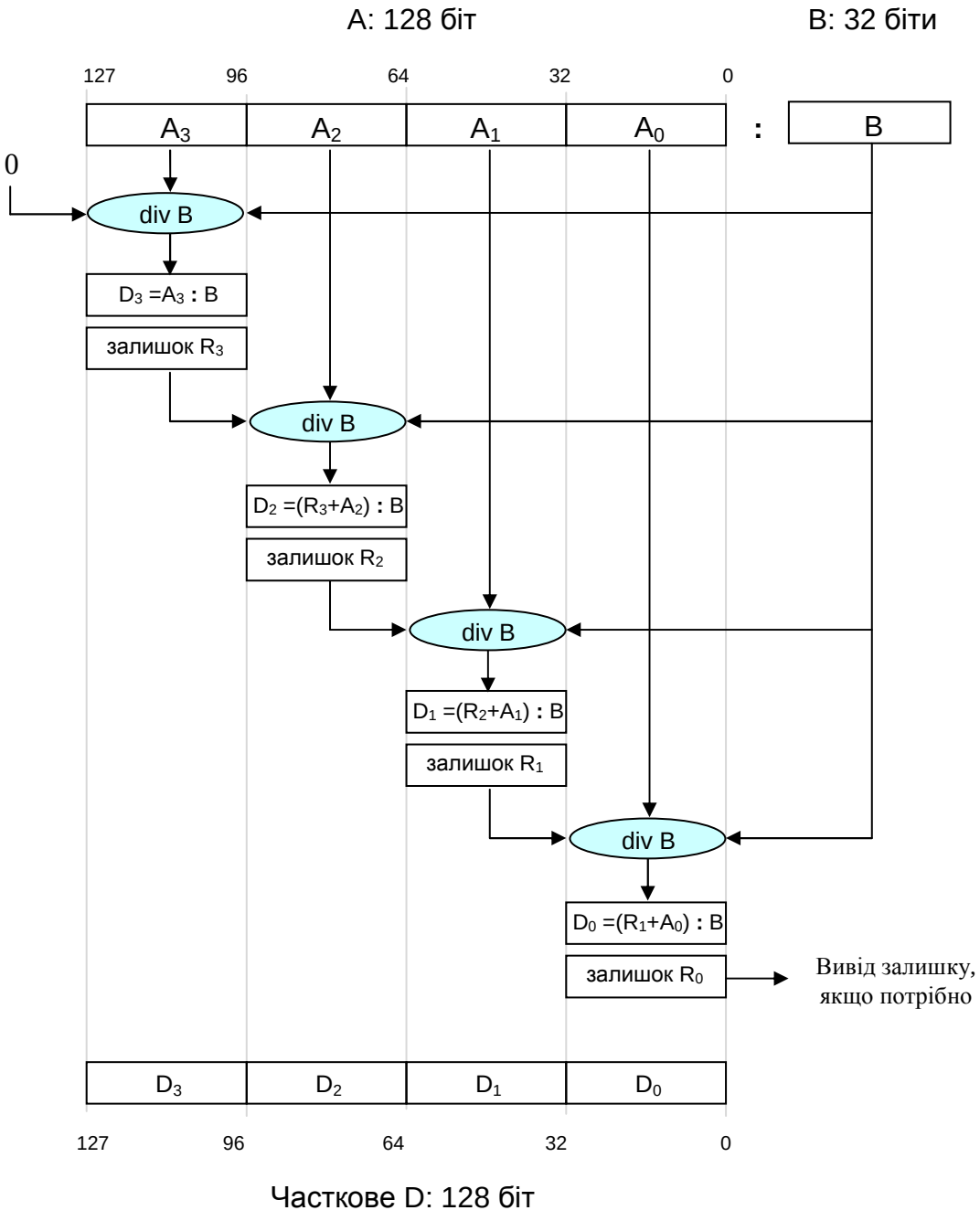


Рис. 3. Приклад ділення групами бітів "128:32"

Команди процесорів x86 для цілочисельного ділення

Команда DIV. Ця команда виконує ділення цілих чисел без знаку: A/B. Синтаксис запису:

```
div операнд
```

Для цієї команди записується тільки один операнд – дільник (*B*), який може бути 8-, 16-, 32- або 64-бітовим. Виконання команди DIV визначається розрядністю цього операнду. У залежності від розрядності операнду *B*, у якості діленого (*A*) буде: або вміст регістру AX, або вміст пари регістрів DX:AX, або вміст пари регістрів EDX:EAX, або вміст пари регістрів RDX:RAX.

Таблиця 1

Розміри операндів A/B	Неявний операнд (ділене A)	Операнд команди (дільник B)	Результати виконання команди DIV		
			Часткове	Залишок	Макс. значення часткового
16/8	AX	r/m8	AL	AH	2^8-1 = 255
32/16	DX:AX	r/m16	AX	DX	2^{16}-1 = 65535
64/32	EDX:EAX	r/m32	EAX	EDX	2^{32}-1
128/64	RDX:RAX	r/m64	RAX	RDX	2^{64}-1

Приклад. Поділити число *A*=60324 на *B*=2014. Дільник *B* буде 16-бітовим операндом команди DIV. Тоді для діленого потрібно використати пару регістрів DX:AX. Відповідний програмний код:

```
xor dx, dx        ;обнулюємо старші 16 бітів діленого
mov ax, 60324     ;молодші 16-бітів діленого
mov bx, 2014      ;16-бітовий дільник
div bx           ;ділення DX:AX / BX
```

Результат: часткове у регістрі AX = 29, залишок у регістрі DX = 1918.

При програмуванні ділення командою DIV необхідно враховувати наступне:

- не можна ділити на нуль;
- обмеження максимальних значень результату, наведені вище у таблиці.

У цих випадках буде згенероване відповідне виключення (*exception*) і програма аварійно завершиться. Наприклад:

```
mov dx, 1          ;старші 16 бітів діленого
mov ax, 0FFFFh     ;молодші 16 бітів діленого
mov bx, 1          ;16-бітовий дільник
div bx     ;спроба ділення DX:AX / BX = 1FFFF / 1 = 1FFFF ?
```

при виконанні команди DIV тут виникне помилка переповнення, оскільки результат 1FFFF не є 16-бітовим.

На відміну цього, наступний код виконується коректно:

```
xor edx, edx          ;обнулюємо старші 32 бітів діленого
mov eax, 1FFFFh       ;молодші 32 бітів діленого
mov ebx, 1            ;32-бітовий дільник
div ebx         ;ділення EDX:EAX / EBX = 1FFFF / 1 = 1FFFF
```

Результати такого ділення:

- часткове (1FFFF) записується у регістр EAX;
- залишок (0) записується у регістр EDX.

Команда IDIV. Ця команда виконує ділення цілих чисел зі знаком: A/B. Синтаксис запису:

```
idiv операнд
```

Так само, як і для команди DIV, для команди IDIV явно вказується тільки один операнд – дільник (*B*). Виконання команди IDIV визначається розрядністю цього операнду (табл.2).

Таблиця 2

Розміри операндів A/B	Неявний операнд (ділене A)	Операнд команди (дільник B)	Результати виконання команди IDIV		
			Часткове	Залишок	Діапазон значень часткового
16/8	AX	r/m8	AL	AH	-128 . . . +127
32/16	DX:AX	r/m16	AX	DX	-32768 . . . +32768
64/32	EDX:EAX	r/m32	EAX	EDX	-2^{31} . . . $2^{31}-1$
128/64	RDX:RAX	r/m64	RAX	RDX	-2^{63} . . . $2^{63}-1$

Якщо результат ділення від'ємний, то часткове та залишок записуються як від'ємні числа у додатковому коді відповідної розрядності.

Приклад. Поділити число *A*= -100000 на *B*=2014. Для представлення числа *A* потрібно як мінімум 32-бітів додаткового коду: *A* = FFFE7960. Число *B* може бути представлене 16-бітовим кодом. Можна вказати, що взагалі не варто писати програмний код для ділення констант, проте для ілюстрації роботи команди IDIV можна записати наступний програмний код:

```
mov dx, 0FFFEh    ;старші 16 бітів діленого A
mov ax, 7960h     ;молодші 16 бітів A
mov cx, 2014      ;дільник B
idiv cx           ;ділення DX:AX / CX = FFFE 7960 / 2014
```

У результати виконання команди IDIV

- часткове FFCF (-49) записується у регістр AX;
- залишок FADE (-1314) записується у регістр DX.

Можна поставити запитання: як запрограмувати ділення перемінних, наданих у 32-бітовому форматі? Це можна зробити, наприклад, так:

```
.data
varA dd -100000                 ;FFFE7960
varB dd 2014                    ;000007DE

.code
mov eax, dword ptr [varA]       ;EAX = FFFE7960
cdq                             ;EDX = FFFFFFFF
mov ecx, dword ptr [varB]       ;ECX = 000007DE
idiv ecx                        ;ділення EDX:EAX / ECX
```

У цьому коді використано команду CDQ, яка перетворює подвійне слово зі знаком, записане у регістрі EAX, у квадрослово EDX:EAX. Значення старшого (знакового) біту регістру EAX розмножується і записується у регістрі EDX. Для від'ємного числа у регістрі EDX усі 32 біти будуть 1.

У результати виконання ділення для наведеного вище коду:
- часткове FFFFFFCF (-49) записується у регістр EAX;
- залишок FFFFFADE (-1314) записується у регістр EDX.

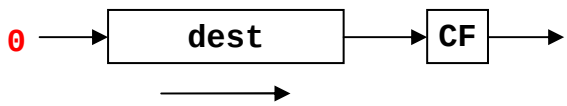
Команди зсуву та їхнє використання у арифметичних операціях

Такі команди зсувають біти двійкового коду операндів.

Команда SHR. Зсув бітів вправо (у напрямку молодшого біту).

```
shr dest, count
```

Виконується зсув бітів коду, записаного у операнді **dest**, вправо на **count** бітів. У старші **count** бітів записуються нулі.



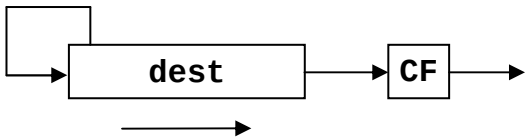
Операнд **dest** може вказувати регістр або адресу пам’яті. Операнд **count** може бути безпосереднім значенням або регістром CL. Значення операнду **count** у 32-бітовому режимі процесора може бути від 0 до 31.

Наприклад:

```
mov eax, 0F00B000h        ;00001111000000001011000000000000 = 251703296
shr eax, 7                ;00000000000011110000000101100000 = 1966432
```

Зсув вправо на 7 бітів дає той самий результат, що й ділення без знаку на 2^7, тобто 1966432 = 251703296 / 128. Таким чином, якщо потрібно запрограмувати спрощенне ділення цілих чисел без знаку на степінь 2, то замість команди DIV можна використовувати SHR, яка працює набагато швидше.

Команда SAR. Зсув бітів вправо арифметичний. Старший (лівий) біт розмножується.



Якщо лівий біт операнду **dest** дорівнює 0, то зсув виконується так само, як і для команди SHR. Наприклад

```
mov eax, 0F00B000h        ;00001111000000001011000000000000
sar eax, 7                ;00000000000111100000000101100000
```

Проте, якщо старший біт операнду є 1, то при зсуві розмножуються вже одиниці:

```
mov eax, -1580246784      ;10100001110011110101100100000000
sar eax, 7                ;11111111010000111001111010110010
```

Команда SAR подібна діленню цілих зі знаком на степінь 2. У наведеному вище прикладі -1580246784 / 128 = -12345678. Виникає питання: якщо дільник степінь 2, то чи можна у програмах ділення використовувати SAR замість IDIV? Для порівняння можна розглянути код на основі команди IDIV:

```
mov eax, -1580246784
cdq
mov ecx, 128
idiv ecx          ; результат: EAX = FF439EB2 (-12345678), EDX = 00000000
```

тут команда IDIV записує у регістр EAX той самий результат, як і для наведеного вище коду для команди SAR.
Розглянемо інший приклад – ділення (-3/4)

```
mov eax, -3
cdq
mov ecx, 4
idiv ecx          ; результат: EAX = 00000000, EDX = FFFFFFFD (-3)
```

Команда IDIV дає результат: (-3/4) = 0, залишок (-3).

А тепер запишемо ділення (-3/4) на основі команди SAR

```
mov eax, -3      ; FFFFFFFD
sar eax, 2       ; результат: EAX = FFFFFFFF (-1)
```

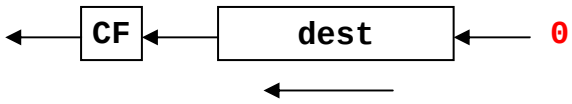
Команда SAR дає інший результат: (-3/4) = -1. На відміну від IDIV, після арифметичного зсуву SAR отримаємо результат, округлений до -1.

Таким чином, програмісту у кожному конкретному випадку треба ретельно вивчати, чи коректно заради прискорення замість ділення використовувати зсув.

Команди SAL, SHL. Зсув бітів вліво (у напрямку старшого біту).

```
shl dest, count
```

Виконується зсув бітів коду, записаного у операнді **dest**, вліво на **count** бітів. У молодші **count** бітів записуються нулі.



Операнд **dest** може вказувати регістр або адресу пам'яті. Операнд **count** може бути безпосереднім значенням або регістром CL. Значення операнду **count** у 32-бітовому режимі процесора може бути від 0 до 31.

Наприклад:

```
mov ax, 2Fh          ; 0000 0000 0010 1111
mov cl, 7
shl ax, cl           ; 0001 0111 1000 0000

mov eax, 2Fh         ; 0000 0000 0000 0000 0000 0000 0010 1111
shl eax, 13          ; 0000 0000 0000 0101 1110 0000 0000 0000
```

Команди SAL та SHL працюють однаково. Наприклад

```
mov eax, 2Fh         ; 0000 0000 0000 0000 0000 0000 0010 1111
sal eax, 13          ; 0000 0000 0000 0101 1110 0000 0000 0000
```

команда SAL записує у операнд призначення та регістр EFLAGS такі самі бітові значення, що й SHL.

Перетворення з двійкової у десяткову систему числення

Одним з відомих алгоритмів перетворення чисел у іншу позиційну систему числення є цілочисельне ділення на основу нової системи. Після кожного ділення береться залишок – він буде цифрою результату. Послідовне ділення продовжується доти результат ділення не стане нулем.

Наприклад, двійкове число 110101110101 переведемо у десяткову систему послідовним діленням на 10.

110101110101 : 1010 = 101011000 залишок 101 (десяткова цифра **5**)
101011000 : 1010 = 100010 залишок 100 (десяткова цифра **4**)
100010 : 1010 = 11 залишок 100 (десяткова цифра **4**)
11 : 1010 = 0 залишок 11 (десяткова цифра **3**)

Результат 3445_{10}

Вивід результатів у десятковій системі числення

Для того, щоб вивести якесь числове значення за допомогою процедури, яка відображає текст, потрібно сформувати відповідний рядок символів, який записати у масив – буфер рядка тексту. Потрібно обробити десяткові цифри числа, перетворивши кожну цифру у однобайтовий код ASCII. Для символів цифр 0, 1, 2, . . , 9 відповідні коди ASCII будуть 48, 49, 50, . . , 57. Таким чином

код ASCII = цифра + 48

Це співвідношення можна використати при програмуванні циклу перетворення числового коду у рядок тексту з десятковими цифрами.

Порядок виконання роботи та методичні рекомендації

1. Створіть у середовищі MS Visual Studio новий проект з ім’ям **Lab7**.

2. Додайте у проект порожній файл з ім’ям **main7.asm**. Цей файл буде головним файлом програмного коду. Для спрощення виконання роботи скористайтеся текстом головного файлу *.asm попередньої роботи №5. Скопіюйте текст і у вікні редагування вихідного тексту вилучіть зайві рядки. Запишіть на диск головний файл програми **main7.asm**.

3. Додайте у проект модулі **module** та **longop**. У проекті використовується файли **module.asm, module.inc, longop.asm**, **longop.inc** попередньої роботи №6.

4. Запрограмуйте процедуру ділення чисел підвищеної розрядності на 10. Назвіть процедуру ім’ям, наприклад, **Div10**. Рекомендується надати таке ім’я, яке позначає не тільки функцію, а й приналежність модулю, у якому ця процедура міститься, наприклад, **Div10_LONGOP**. У процедури повинні бути такі аргументи: адреса числа підвищеної розрядності, розрядність (кількість бітів) та адреси результату – часткового та залишку від ділення на 10.

5. Запрограмуйте процедуру перетворення у десятковий код чисел підвищеної розрядності. Назвіть процедуру **StrDec**. У процедурі повинні бути три аргументи: адреса буфера тексту-результату, адреса числового значення, кількість бітів числа (параметри такі самі, як у процедури StrHex_MY). Для перетворення у десятковий код використати послідовне ділення на 10. Рекомендується для ділення на 10 викликати процедуру Div10, яка запрограмована у п. 4 даної роботи.

6. Потрібно вирішити, у яких модулях будуть розташовуватися програмні коди нових процедури Div10 та StrDec.

7. У файлі **main7.asm** потрібно запрограмувати цикл для обчислення значення факторіалу – рекомендується скористатися кодом попередньої роботи №5. Для виводу десяткового значення факторіалу треба викликати процедуру StrDec.

8. Також у файлі **main7.asm** потрібно запрограмувати обчислення значення вираження за формулою згідно варіанту завдання.

9. Запрограмувати вивід результатів у діалоговому вікні MessageBox. Запрограмувати вивід потрібних числових значень у десятковому коді.

10. Компіляція, виклик програми, налагодження, отримання результатів. Виконання цих дій виконується у середовищі MS Visual Studio. Відомості та методичні рекомендації надані у відповідних розділах попередніх робіт.

Зміст звіту:

1. Титульний лист
2. Завдання
3. Роздруківка тексту програми
4. Роздруківка результатів виконання програми
5. Аналіз, коментар результатів, вихідного тексту та дизасембльованого машинного коду
6. Висновки

Варіанти завдання

1. Потрібно запрограмувати на асемблері вивід значення факторіалу *n*! у **десятковому коді**. Використати програмний код обчислення факторіалу попередньої лабораторної роботи №5. Для кожного студента своє значення *n*

$n = 30 + 2 \times H$,

де *H* – це номер студента у журналі.

2. Перетворення у десятковий код запрограмувати діленням числа підвищеної розрядності на 10. Ділення на 10 виконати двома способами – діленням "у стовпчик" та діленням груп бітів. Ділення групами бітів запрограмувати або як 8-бітове, або як 16-бітове, або як 32-бітове відповідно до варіанту. Варіант ділення групами бітів обирається відповідно залишку ділення номеру студента у журналі (*H)* на 3:

- залишок = 0 – ділення групами по 8 бітів;

- залишок = 1 – ділення групами по 16 бітів;

- залишок = 2 – ділення групами по 32 бітів.

3. Програмний код ділення "у стовпчик" та ділення групами бітів оформити у вигляді процедур. Процедури повинні містити такі параметри: адреса ділимого, розрядність ділимого, значення дільника (B) та інші (за необхідності).

4. Запрограмувати на асемблері також обчислення формули, яку вибрати з таблиці, наведеної нижче. Номер варіанту формули згідно номеру в журналі.

Формула має вигляд

$y = F(x, m)$

Значення *x*, *y* повинні бути 32-бітовими цілими зі знаком. Значення *m* – ціле без знаку. Результат (*y*) повинен записуватися у регістр EAX.

Варіанти формул $y = F(x, m)$

Номер	Формула	Номер	Формула	Номер	Формула
1	$y = \frac{x}{x+1} 2^{m}$	11	$y = \frac{x}{x+3} 2^{m+2}$	21	$y = \frac{x}{x+5} 2^{m+4}$
2	$y = \frac{x-1}{x+1} 2^{m}$	12	$y = \frac{x-3}{x+3} 2^{m-2}$	22	$y = \frac{x-5}{x+5} 2^{m-4}$
3	$y = \frac{x}{3} 2^{m}$	13	$y = \frac{x}{7} 2^{m+2}$	23	$y = \frac{x}{11} 2^{m+4}$
4	$y = \frac{3}{x+1} 2^{m}$	14	$y = \frac{7}{x+1} 2^{m}$	24	$y = \frac{11}{x+1} 2^{m}$
5	$y = \frac{x}{3} 2^{-m}$	15	$y = \frac{x}{7} 2^{-m}$	25	$y = \frac{x}{11} 2^{-m}$
6	$y = \frac{x}{x+2} 2^{m+1}$	16	$y = \frac{x}{x+4} 2^{m+3}$	26	$y = \frac{x}{x+6} 2^{m+5}$
7	$y = \frac{x-2}{x+2} 2^{m-1}$	17	$y = \frac{x-4}{x+4} 2^{m-3}$	27	$y = \frac{x-6}{x+6} 2^{m-5}$
8	$y = \frac{x}{5} 2^{m+1}$	18	$y = \frac{x}{9} 2^{m+3}$	28	$y = \frac{x}{13} 2^{m+5}$
9	$y = \frac{5}{x+1} 2^{m}$	19	$y = \frac{9}{x+1} 2^{m}$	29	$y = \frac{13}{x+1} 2^{m}$
10	$y = \frac{x}{5} 2^{-m}$	20	$y = \frac{x}{9} 2^{-m}$	30	$y = \frac{x}{13} 2^{-m}$

Контрольні питання:

1. Що таке цілочисельне ділення?
2. Як виконується команда DIV?
3. Як виконується команда IDIV?
4. Коли можна замість ділення робити зсув?
5. Чим відрізняється команда SAR від SHR?
6. Як виконується ділення чисел підвищеної розрядності?
7. Як перевести з двійкової у десяткову систему?